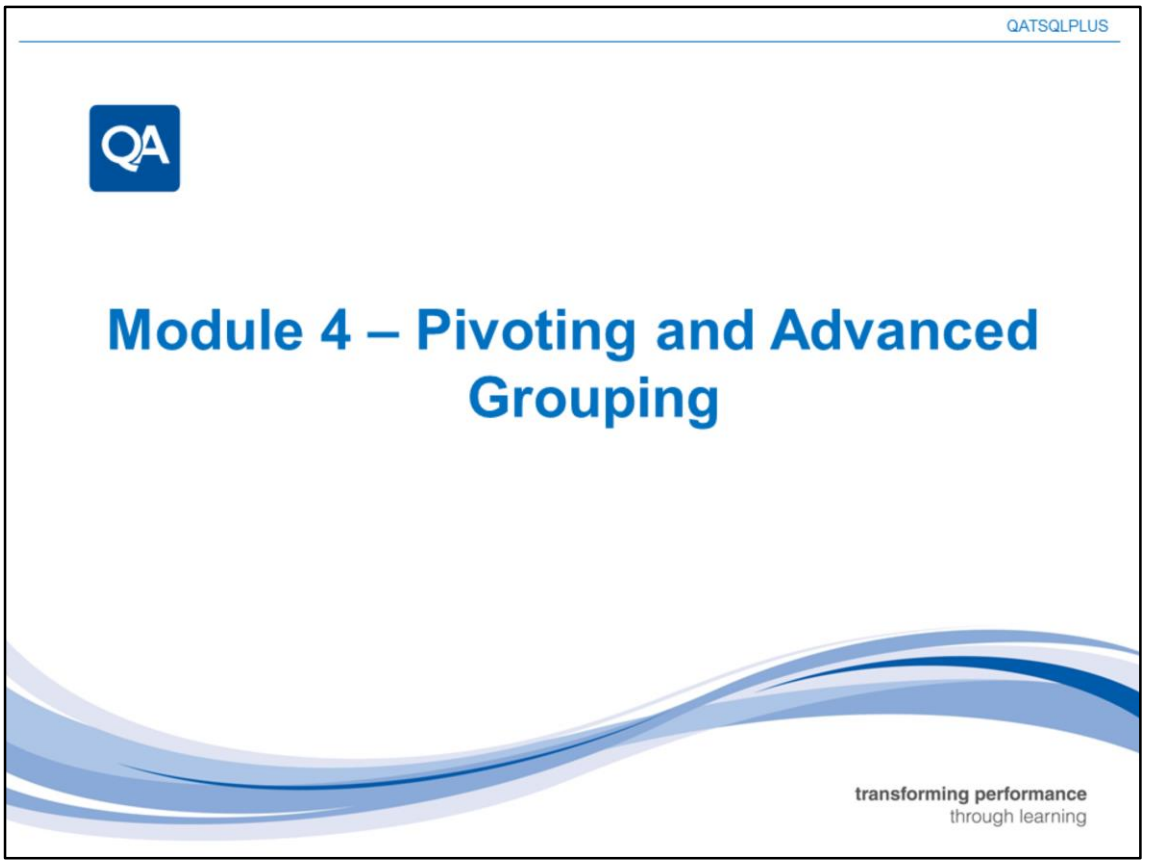
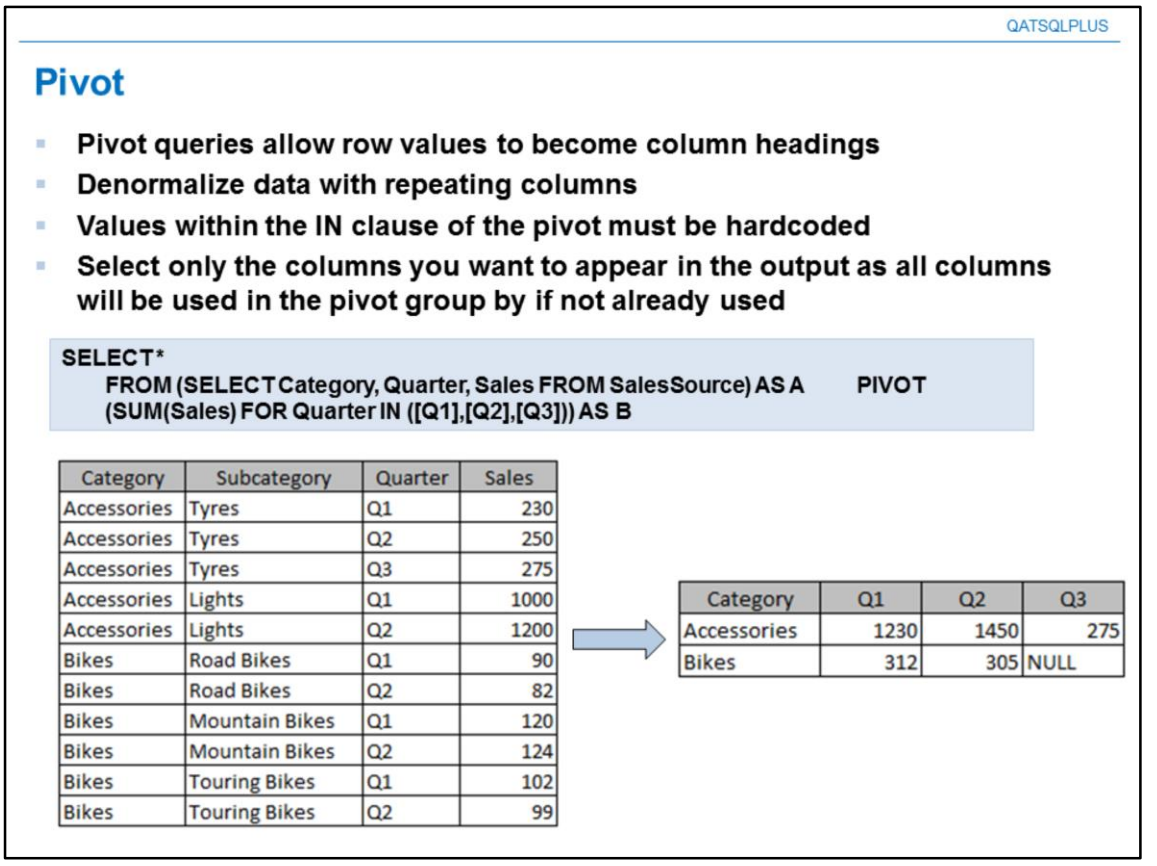

QATSQLPLUS
Module 4 – Pivoting and Advanced Grouping
transforming performance
through learning

QATSQLPLUS

Alternative Analysis

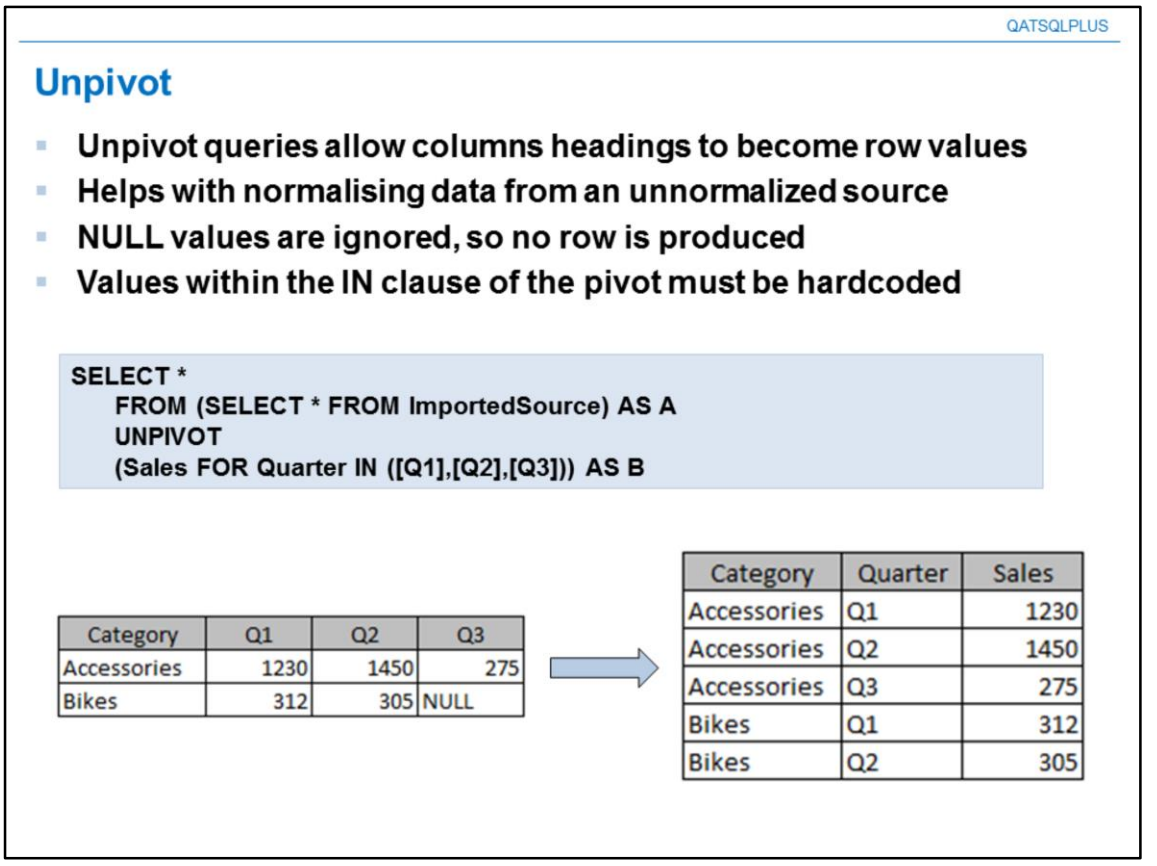
- **Pivot**
- **Unpivot**
- **Adding summary rows**
  - Rollup
  - Cube
  - Grouping sets



Pivot

- Pivot queries allow row values to become column headings, although each value in the pivoted column needs to be explicitly listed
    - Dynamic SQL can be used to dynamically build the values list, although this may cause issues with the client tools accessing the dataset
- Denormalize data with repeating columns, where the output requires the value in the columns
    - This can usually be performed in the presentation layer such as Visual Studio app or SSRS report
- Select only the columns you want to appear in the output as all columns will be used in the pivot group by if not already used

In the example on the slide above, the subcategory is not required in the output dataset so should not be selected in the first derived table. The Quarter values that are required as column headings are Q1, Q2 and Q3. If Q4 was in the original dataset then it would not be returned in the pivoted dataset. The Sales column is used in the aggregation at the start of the pivot clause.



Unpivot

- Unpivot queries allow columns headings to become row values which helps with normalising data from an unnormalized source
- NULL values are ignored in the source data values, so no row is produced
- Values within the IN clause of the pivot must be hardcoded, although a dynamic SQL statement can be built if necessary
    - Unlike the pivot statement, this does not change the output structure so would not cause issues with client tools

Example

A common query may be to return any quarter where the sales are greater than 1000. An example of such a query appears below. This would return any row where any of the columns match the criteria. It would not specify the quarter.

```
SELECT *
    FROM ImportedSource
    WHERE Q1 > 1000 OR Q2 > 1000 OR Q3 > 1000
```

If pivoted, then the query was used in a CTE, the query could be as below. The returned values would be the individual quarters for a category where the criteria is met.

```
WITH QuarterSales AS (
    SELECT *
        FROM
        (SELECT Category, Q1, Q2, Q3 FROM Imported Source) AS RawData
        PIVOT
        (Sales FOR Quarter IN ([Q1],[Q2],[Q3])) AS Unpivotted
)
SELECT *
    FROM QuarterSales
    WHERE Sales > 1000
```

QATSQLPLUS

Rollup

- **Adds additional grouped rows progressively for each level named in GROUP BY clause**
- **Lower levels are summarised and shown as NULL**

```
SELECT Category, Quarter, SUM(Sales) AS TotalSales
    FROM SalesSource
    GROUP BY Category, Quarter
        WITH ROLLUP
```

Category	Subcategory	Quarter	Sales
Accessories	Tyres	Q1	230
Accessories	Tyres	Q2	250
Accessories	Tyres	Q3	275
Accessories	Lights	Q1	1000
Accessories	Lights	Q2	1200
Bikes	Road Bikes	Q1	90
Bikes	Road Bikes	Q2	82
Bikes	Mountain Bikes	Q1	120
Bikes	Mountain Bikes	Q2	124
Bikes	Touring Bikes	Q1	102
Bikes	Touring Bikes	Q2	99

Category	Quarter	Sales
Accessories	Q1	1230
Accessories	Q2	1450
Accessories	Q3	275
Accessories	NULL	2955
Bikes	Q1	312
Bikes	Q2	305
Bikes	NULL	617
NULL	NULL	3572

Rollup

- Rollup adds additional rows for the groups listed in the GROUP BY clause
- Working from right to left in the GROUP BY would summarise the remaining group by columns regardless of the value in the current column
- The summary rows are shown as NULL for the ignored columns
- This process is normally handled by the client tool rather than using ROLLUP

The query

```
SELECT Category, Quarter, SUM(Sales) AS TotalSales
    FROM SalesSource
    GROUP BY Category, Quarter WITH ROLLUP
```

Would produce

The original rows for GROUP BY would return Category and Quarter with a Sales value. The quarter column is then chosen to be ignored, so the Sales value would be the sales for the category as a whole regardless of quarter. Then the category and quarter columns are selected to be ignored, so the result is the Sales for the complete dataset.

1. Sales for combinations of Category and Quarter
2. Sales for Category ignoring Quarter
3. Sales for all

QATSQLPLUS

Cube

- Adds additional grouped rows for combinations of levels named in GROUP BY clause
- Grouped levels are shown as NULL

```
SELECT Category, Quarter, SUM(Sales) AS Sales
    FROM SalesSource
    GROUP BY Category, Quarter
        WITH CUBE
```

Category	Subcategory	Quarter	Sales
Accessories	Tyres	Q1	230
Accessories	Tyres	Q2	250
Accessories	Tyres	Q3	275
Accessories	Lights	Q1	1000
Accessories	Lights	Q2	1200
Bikes	Road Bikes	Q1	90
Bikes	Road Bikes	Q2	82
Bikes	Mountain Bikes	Q1	120
Bikes	Mountain Bikes	Q2	124
Bikes	Touring Bikes	Q1	102
Bikes	Touring Bikes	Q2	99

Category	Quarter	Sales
Accessories	Q1	1230
Accessories	Q2	1450
Accessories	Q3	275
Accessories	NULL	2955
Bikes	Q1	312
Bikes	Q2	305
Bikes	NULL	617
NULL	NULL	3572
NULL	Q1	1542
NULL	Q2	1755
NULL	Q3	275

Cube

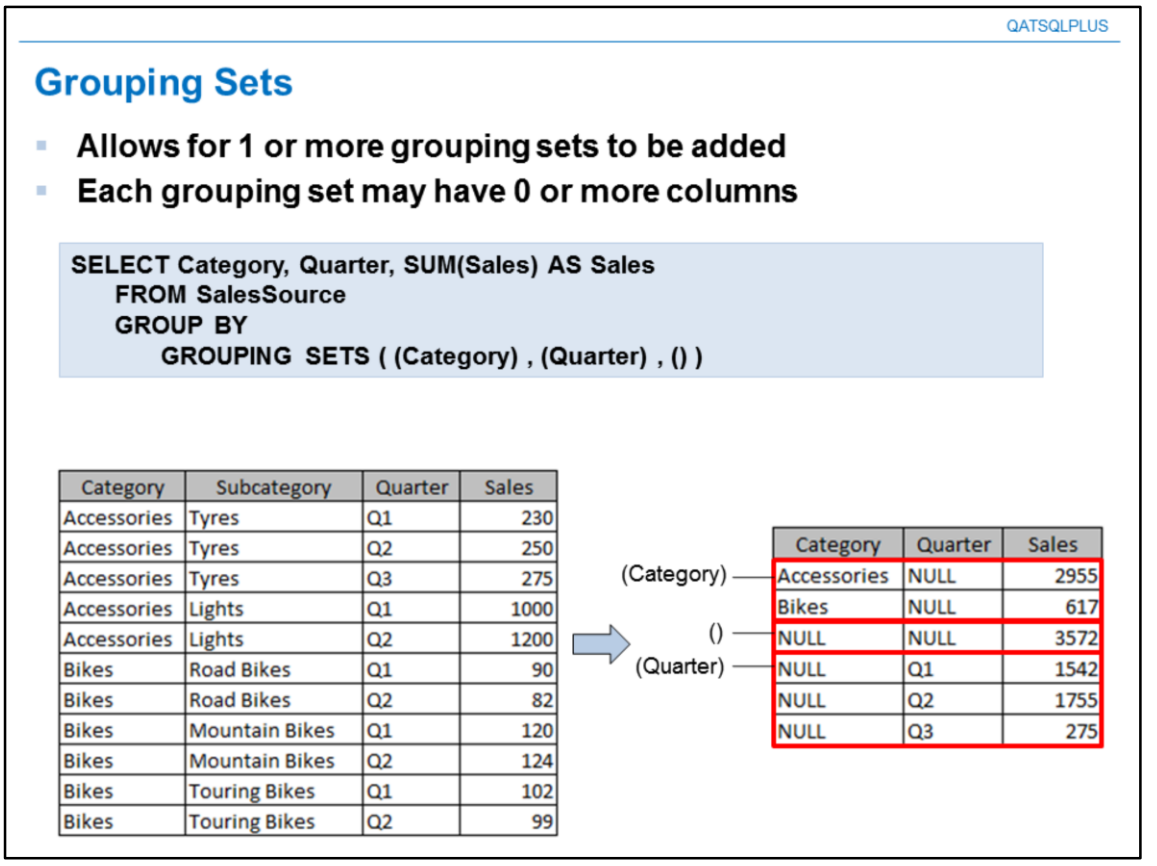
- Cube adds additional rows for the groups listed in the GROUP BY clause
- Working with all combinations of zero or more, columns in the GROUP BY would summarise the select columns regardless of the remaining column in the GROUP BY
- The summary rows are shown as NULL for the ignored columns
- This process is normally handled by the client tool rather than using CUBE, such as a matrix in SSRS

The query

```
SELECT Category, Quarter, SUM(Sales) AS TotalSales
    FROM SalesSource
    GROUP BY Category, Quarter WITH CUBE
```

Would produce

1. Sales for combinations of Category and Quarter
2. Sales for Category ignoring Quarter
3. Sales for Quarter ignoring Category
4. Sales for all



Grouping Sets

- Similar to Rollup and Cube but with more control over the extra summary rows that can be added
- The grouping set holds a group of one or more column combinations
- Each non-aggregate column in the SELECT list must be used at least once in the grouping sets
- In a more complex query, columns A, B, C and D could be grouped, in that order
    - Grouping sets allows for any of the possible combinations, listed below, to be selected
    - The list contains each combination of columns allowed in the grouping set list, including the empty set ( ) for grand total

(A, B, C, D), (A, B, C),(A, B, D),(A, C, D),(B, C, D),(A, B),(A, C),(A, D),(B, C),(B, D),(C, D),(A),(B),(C),(D),( )

Example

The query below will return the original grouped rows (VendorName, CourseName and StartDate), the total for the (VendorName), the total for the (StartDate) and the grand total ( ).

```
SELECT Vendorname, CourseName, StartDate, SUM(NumberDelegates)
   FROM dbo.VendorCourseDateDelegateCount
   GROUP BY
   GROUPING SETS (
      (VendorName, CourseName, StartDate),
      (VendorName),
      (StartDate),
      ( )
   )
```

QATSQLPLUS

Grouping

- **Aggregate function that shows aggregated rows when using rollup, cube and grouping sets**
- **Parameter is the column name that is grouped or not in this row**
- **Result: 1 = grouped, 0 = not grouped**

```
SELECT Category, Grouping(Category) AS GC, Quarter, Grouping(Quarter) AS GQ,
SUM(Sales) AS Sales
    FROM SalesSource
    GROUP BY Category, Quarter
        WITH CUBE
```

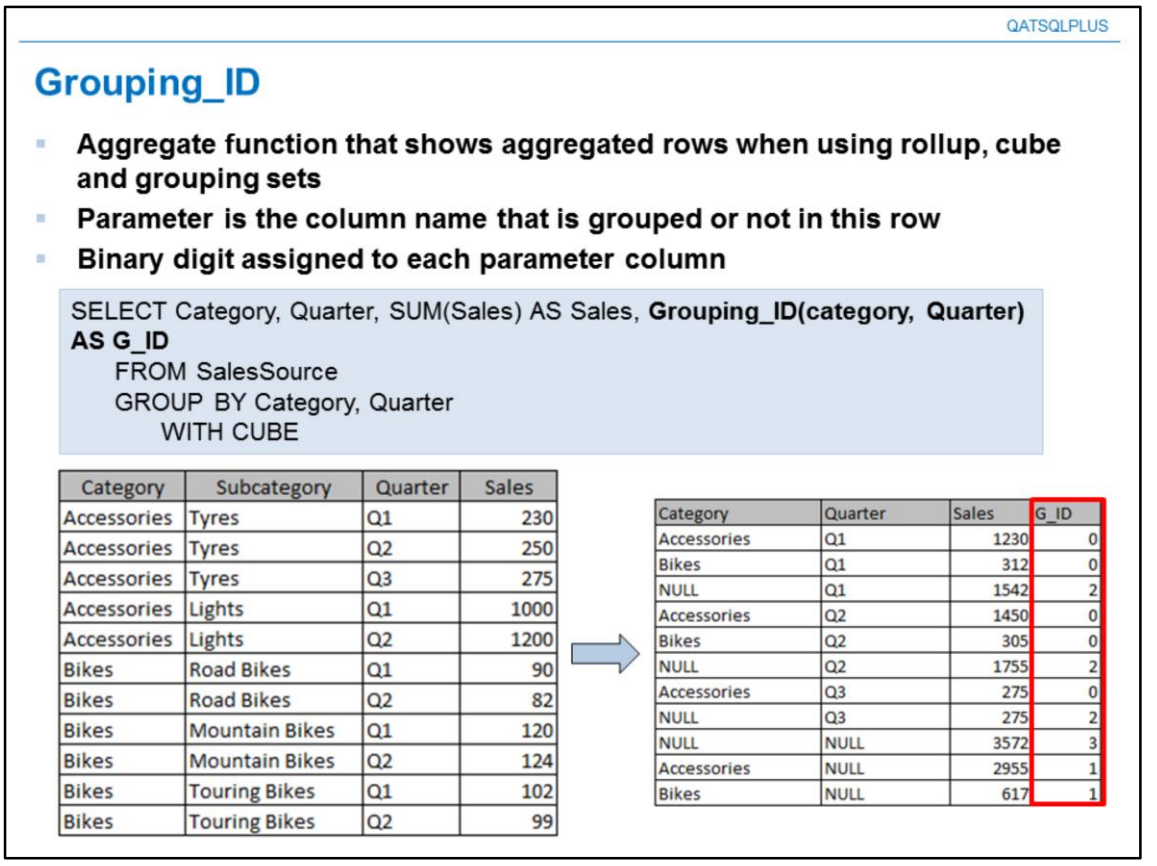
Category	Subcategory	Quarter	Sales
Accessories	Tyres	Q1	230
Accessories	Tyres	Q2	250
Accessories	Tyres	Q3	275
Accessories	Lights	Q1	1000
Accessories	Lights	Q2	1200
Bikes	Road Bikes	Q1	90
Bikes	Road Bikes	Q2	82
Bikes	Mountain Bikes	Q1	120
Bikes	Mountain Bikes	Q2	124
Bikes	Touring Bikes	Q1	102
Bikes	Touring Bikes	Q2	99

Category	GC	Quarter	GQ	Sales
Accessories	0	Q1	0	1230
Bikes	0	Q1	0	312
NULL	1	Q1	0	1542
Accessories	0	Q2	0	1450
Bikes	0	Q2	0	305
NULL	1	Q2	0	1755
Accessories	0	Q3	0	275
NULL	1	Q3	0	275
NULL	1	NULL	1	3572
Accessories	0	NULL	1	2955
Bikes	0	NULL	1	617

Grouping

- In some datasets the group by column may have a NULL value
    - In this case a row that shows a NULL may be a grouped row or a summary row
- The grouping function returns a 0 or 1 dependent upon whether the column passed is a NULL for a summary row or not

In the example in the slide, the summary rows have a 1 in the GC or GQ column.



Grouping_ID

- The Grouping_ID function is similar to Grouping function, but can take multiple columns as input
- Each column is allocated a binary power value from right to left (16, 8, 4, 2, 1, etc.)
- If a column is a grouped column (grouping = 1) then the assigned value is added to the Grouping_ID value

In the slide example, Quarter = 1 and Category = 2.

Grouping(Category, Quarter) column G_ID values

0 Neither column summarised

1 Quarter summarised only

2 Category summarised only

3 Both columns summarised

QATSQLPLUS

Exercise